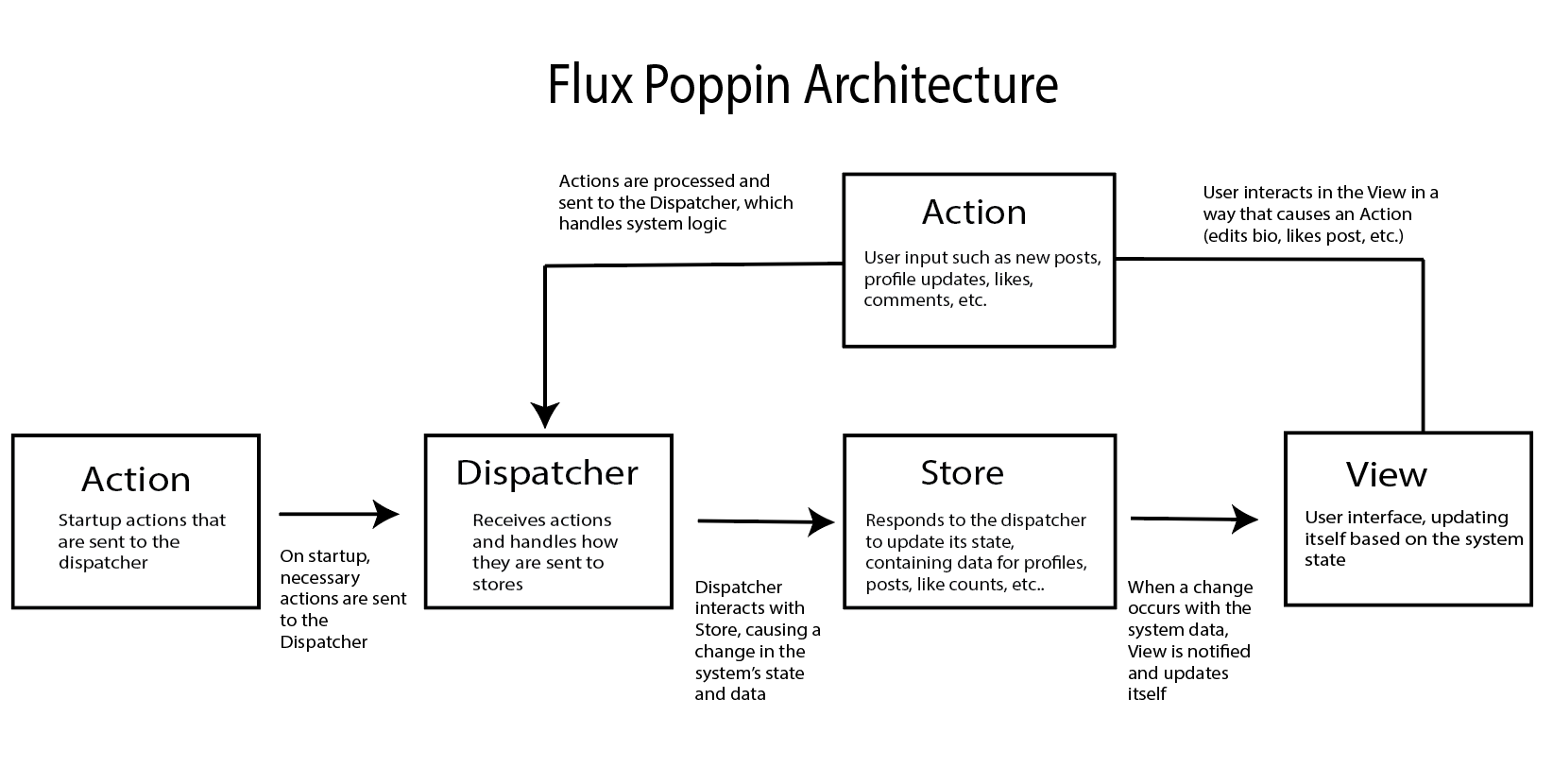
Software Architecture for WEBMB’s Poppin

by [Monica Agneta](mailto:monica.agneta@maine.edu) and Brett Palmer

**Justification)**

Because Poppin is a social media site that is centered around user interactions and shared media, Flux was the best system architecture to choose. In Poppin, the main experience of the app is through features such as posting, liking, and commenting. First, the user interacts with the UI, such as following a user. This interaction can be seen as an action. Next, business logic dispatches the actions to determine how they should impact system data — for example, following a user should increase their following count. Lastly, a change in system data updates the UI for the user to know that their follow action was successful. The series of actions just described is essential to Poppin and it follows the basic structure of Flux architecture.

Overall, the unidirectional data flow of Flux efficiently handles user interactions that update the system state in real-time, which is a core requirement for Poppin. Also, Flux’s clear separation of responsibilities also fits well with Poppin’s diverse features. Each component of Flux has a specific job, making the code easier to maintain as Poppin is expanded with more features. Lastly, Facebook, a popular social media platform whose functionality is similar to Poppin, successfully uses Flux for its architecture. For all of these reasons, WEBMB has decided to develop Poppin based on Flux principles.

**Components)**

Action:

Actions are events that are triggered by user interaction in the view. There are different types of actions, because there are different types of user interactions. For instance, a user clicking an “Upload Post” button triggers a type of action for uploading a post. Additionally, an action could be something that occurs without user interaction, such as a data backup. Each action is an event that influences the state of the system. However, for an action to impact the system state, it must be processed by the dispatcher.

Dispatcher:

Once an action occurs, it is sent to the dispatcher which acts as a central hub for all of the user actions. The dispatcher receives incoming actions and distributes them to the appropriate stores for processing. It does this by containing logic that builds associations between actions types and one or more stores. By acting as a traffic controller that directs change from actions to stores, the dispatcher maintains a unidirectional flow of data.

Store:

The dispatcher interacts with the store component which is split into various stores that hold data specific to certain application states. Examples of stores are user data, posts, and profile. All stores receive the action from the dispatcher and they determine if they need to respond back by updating accordingly. The stores decision to change based on actions is declared by how the stores originally register with the dispatcher component before the application runs. If a store is determined to be involved with a certain action that is dispatched to it, it will apply changes to its state, typically by altering data held within the store. Examples of data updated are things such as added followers, update to media on profile, or a new comment on a post.

View:

Once the stores have updated accordingly, the view component is notified of a changed state. The view component is made up of React components such as list items, tabs, or cards. The components can update themselves upon receiving a notification of a change in state related to their functionality. Then, the user interface will update to reflect what is new in the application state. This allows the user to view new changes in Poppin via the UI. Also, some updates to view may prompt the user for more action. For example, if the original user action was clicking the “Update Profile Bio” button, the user interface would update to present an editing box. This view update relies on another user action of typing an update profile bio. These resulting actions would then feed back into the dispatcher component and begin another unidirectional flow through the flux system.